ABDELLAH SALHI
GEORGE R. LINDFIELD

Effects of ordering and updating techniques on the performance of the Karmarkar algorithm

RAIRO. Recherche opérationnelle, tome 25, nº 2 (1991), p. 209-235
<http://www.numdam.org/item?id=RO_1991__25_2_209_0>

Recherche opérationnelle/Operations Research
(vol. 25, n° 2, 1991, p. 209 à 236)

EFFECTS OF ORDERING AND UPDATING TECHNIQUES ON THE PERFORMANCE OF THE KARMARKAR ALGORITHM (*)

by Abdellah SALHI (1) and George R. LINDFIELD (1)

Abstract. — *This paper will report experimental results obtained with two implementations of the Karmarkar algorithm. In one implementation it is assumed that the optimum objective value is available while in the other, this assumption is dropped. The study is particularly concerned with the performance of the algorithm in conjunction with sparsity preservation techniques, namely the nested dissection ordering algorithm of Alan George and the updating algorithm for least squares of Heath. Standard test problems are solved with FORTRAN 77 codes of the algorithm as well as with the well known LP package LINDO. In addition, a numerical example is given to illustrate the algorithm and its geometric features.*

Résumé. — *Cet article présente des résultats expérimentaux obtenus avec deux programmes de l'algorithme de Karmarkar. Dans l'un, il a été présumé que la valeur optimale de la fonction objective est* a priori *connue, alors que dans l'autre cette supposition est rejetée. L'étude concerne particulièrement la performance de l'algorithme en relation avec les techniques de préservation de l'éparpillement, telles que la méthode de classement d'Alan George et la méthode de mise à jour de la solution du problème des moindres carrés. Les deux programmes, écrits en FORTRAN 77, ont été testés sur des problèmes de programmation linéaire. Ces mêmes problèmes ont été aussi résolus avec le package LINDO.*

1. INTRODUCTION

The algorithm of Karmarkar (1984*a*, 1984*b*) for linear programming was developed as a result of the search for a method which had polynomial complexity like the ellipsoid algorithm but provided a practical implementation like the simplex method. It is related to classical interior point methods, but presents original features such as the use of projective geometry and a logarithmic potential function to measure convergence.

Moving in the direction of the gradient is the natural choice for obtaining an improvement in the objective function when interior point methods are

(*) Received mai 1990.

(1) Department of Computer Science and Applied Mathematics, Aston University, Birmingham B4 7ET, England.

considered for linear programming. However, this will yield a substantial improvement in the objective function only if the current feasible point is at the centre of the polytope, *i. e.* sufficiently distant from all its boundaries. Consequently, for an iterative process to work with these ideas, it must alternate between centring the feasible point and taking a step in the gradient direction.

Classical interior methods of the Brown-Koopmans type [Charnes *et al.*, 1984] have difficulties near the boundaries, precisely because they lack the centring step. The difficulties, usually, result in the loss of feasibility and slow convergence. However, Karmarkar's algorithm successfully combines the two steps and thus avoids the difficulties of the classical methods.

The centring process is performed by rescaling the feasible region at each iteration using a projective transformation. The optimization problem is approximated by a minimization over a sphere of known centre and radius. The minimization over the sphere is then solved by taking a step to its boundary along a projected gradient direction. The rescaling process combined with the step along the negative projected gradient is repeated until optimality is achieved or the problem is recognized to be unbounded or infeasible. We now describe the basic features of the algorithm.

2. THE PROJECTIVE ALGORITHM OF KARMARKAR

Consider the linear programming problem in standard form

SLP_x :

$$\begin{gathered}\operatorname{Min} \mathbf{c}^T \mathbf{x}\\ \text{s. t. } A\mathbf{x}=\mathbf{b}\\ \mathbf{x}\geqq 0,\end{gathered}$$

where R^n is the n-dimensional Euclidean space; $\mathbf{x}$, $\mathbf{c}\in R^n$, $\mathbf{b}\in R^m$ and $A\in R^{m\times n}$. The original Karmarkar algorithm requires that the LP problem is expressed in a special form called the canonical form, which is

PC :

$$\begin{gathered}\operatorname{Min} \mathbf{c}^T \mathbf{x}\\ \text{s. t. } A\mathbf{x}=0\\ \mathbf{e}^T\mathbf{x}=1\\ \mathbf{x}\geqq 0\end{gathered}$$

where $\mathbf{e}^T=(1, 1, \ldots, 1)$.

In addition, it is required that the minimum objective value is 0, and the value of the objective at any feasible and nonoptimal point is strictly positive. The question of converting SLP_x into PC will be treated in detail later.

The centring scheme of Karmarkar is based on a projective transformation defined by

$$T_x(\mathbf{x}) = \frac{D^{-1}\mathbf{x}}{\mathbf{e}^T D^{-1}\mathbf{x}} = \mathbf{x}',$$

and its inverse

$$T_x^{-1}(\mathbf{x}') = \frac{D\mathbf{x}'}{\mathbf{e}^T \mathbf{D}\mathbf{x}'} = \mathbf{x},$$

where $D = \operatorname{diag}(\mathbf{x}^{(k)})$, $\mathbf{x}^{(k)}$ being a point in the space of PC.

Transforming PC using T_x^{-1}, results in a nonlinear (fractional) programming problem with the objective function $\mathbf{c}^T\mathbf{D}\mathbf{x}'/\mathbf{e}^T\mathbf{D}\mathbf{x}'$. However, $\mathbf{e}^T\mathbf{D}\mathbf{x}'$ being positive in the transformed feasible region and given that $\mathbf{c}^T\mathbf{x}$ has minimum zero, $\mathbf{c}^T\mathbf{D}\mathbf{x}'/\mathbf{e}^T\mathbf{D}\mathbf{x}'$ **has also minimum zero. Thus, it can be approximated with** $\mathbf{c}^T\mathbf{D}\mathbf{x}'$. **It follows that the transformed problem to be considered is**

$P_{x'}$:

$$\begin{gathered} \operatorname{Min} \mathbf{c}^T\mathbf{D}\mathbf{x}' \\ \text{s.t. } AD\mathbf{x}' = 0 \\ \mathbf{e}^T\mathbf{x}' = 1, \qquad \mathbf{x}' \geqq 0, \end{gathered}$$

which is of the required form PC.

This problem is an optimization over the intersection of the simplex $\Sigma = \{\mathbf{x}' \in R^{n+1} \colon \mathbf{x}' \geqq 0,\ \ \Sigma x'_j = 1\}$, with the linear subspace $\Pi = \{\mathbf{x}' \in R^{n+1} : \mathbf{x}' \geqq 0,\ A\mathbf{x}' = 0\}$. The centre of the simplex $\mathbf{x}_0'^T = (1/(n+1), 1/(n+1), \ldots)$, being a feasible point, a reduction in the objective function is likely to be achieved along the opposite direction of the projected gradient $\mathbf{p}$, starting from $\mathbf{x}'_0$. However, to insure feasibility after the move, Karmarkar considered the minimization over the largest inscribed sphere S_r in Σ, as an approximation to the minimization over the simplex Σ. This insures feasibility of the resulting point. The problem is written

$P_{x'_s}$:

$$\begin{gathered} \operatorname{Min} \mathbf{c}^T\mathbf{D}\mathbf{x}' \\ \text{s.t. } AD\mathbf{x}' = 0 \\ \mathbf{e}^T\mathbf{x}' = 1 \\ \|\mathbf{x}' - (\mathbf{e}/n)\| \leqq \alpha r \\ \mathbf{x}' \geqq 0, \end{gathered}$$

where $r=1/\sqrt{(n(n-1))}$ is the radius of S_r.

From the geometric point of view, there are 3 spaces involved: the space of the original problem, the space of the homogeneous form of the problem and its image resulting from the projective transformation. Call the last two spaces respectively x-space and x'-space. A sketch of the optimization process is given in figure 2.1.

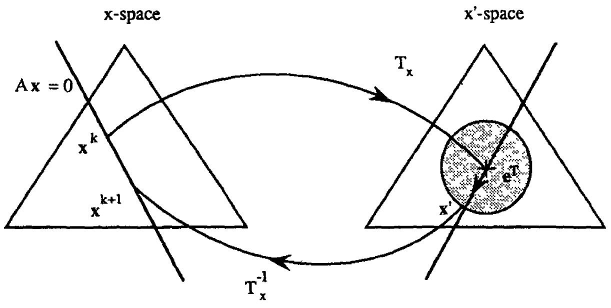


Figure 2.1. – An Iteration of the Algorithm.

Let $\mathbf{x}^{(k)}$ be a point in x-space. Applying the projective transformation T_x to $\mathbf{x}^{(k)}$ results in the centre of S_r in x'-space. A new point in x'-space would be $\mathbf{x}'^{(k)}$ at the boundary of S_r. This point is transformed back into the x-space by the inverse projective transformation T_x^{-1}, resulting in a point $\mathbf{x}^{(k+1)}$. It is easy to see that an improvement in the objective function of $P_{x_s'}$ is achieved in the direction of the projected gradient. The reduction in the objective function of PC is harder to see, when we know that the set of linear functions is not invariant under projective transformations. In this respect, Karmarkar introduced the logarithmic potential function $F(x)=n\ln(\mathbf{c}^T\mathbf{x})-\Sigma_j\ln(x_j)$, where ln is the logarithm of base e, which is invariant under projective transformations. To see that, we write the potential function associated to the objective function in the transformed problem

$$F(\mathbf{x}')=\Sigma_j\ln(\mathbf{c}^T\mathbf{D}\mathbf{x}'/x_j') \qquad (2.1)$$

and in the x-space after applying inverse transformation to $\mathbf{x}'$

$$F(T_x^{-1}(\mathbf{x}'))=\Sigma_j\ln(\mathbf{c}^T\mathbf{D}\mathbf{x}'/x_j')-\Sigma_j\ln x_j. \qquad (2.2)$$

Expressions (2.1) and (2.2) are similar except for a constant $-\Sigma_j\ln x_j$.

Karmarkar proved that a positive constant reduction is achieved in the potential function associated with $\mathbf{c}^T Dx$, when moving from the centre of S_r to its boundary. From (2.1) and (2.2), this reduction corresponds to some reduction in the image of the potential function in x-space. It follows that

$$F(\mathbf{x}^{(k+1)}) \leqq F(\mathbf{x}^{(k)}) - \delta, \tag{2.3}$$

where δ is a positive constant. Based on these considerations, he established the polynomial complexity of the following algorithm.

Algorithm 2.1

Karmarkar's algorithm generates a sequence of points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}, \ldots$ with the assumption that $\mathbf{x}^{(k)} \geqq 0$, $k=1, \ldots$ Assume also that an interior starting point $\mathbf{x}^{(0)}$ is available, and an arbitrarily small value ε is chosen, then the algorithm can be described in the following steps.

0. $k=0$.

1. Set $D=\operatorname{diag}(\mathbf{x}^{(k)})$ and $B=\begin{pmatrix} AD \\ \mathbf{e}^T \end{pmatrix}$.

2. Project vector $D\mathbf{c}$ onto the null space of B to find $\mathbf{p}=HD\mathbf{c}$ where the projection matrix

$$H=I-B^T(BB^T)^{-1}B.$$

3. Normalize $\mathbf{p}$ and scale it by the radius $r=1/\sqrt{(n(n-1))}$ of S_r to find the direction vector $\mathbf{p}'=r\,\mathbf{p}/\|\mathbf{p}\|$.

4. Compute a new feasible point in x'-space by taking a step of length α along $\mathbf{p}'$, starting from the centre $\mathbf{e}/n$ of S_r

$$\mathbf{x}'=\mathbf{e}/n-\alpha\,\mathbf{p}', \qquad \alpha \in (0, 1).$$

5. Apply inverse transformation to $\mathbf{x}'$ to find a new point in x-space

$$\mathbf{x}^{(k+1)}=\frac{D\mathbf{x}'}{\mathbf{e}^T D\mathbf{x}'}.$$

6. Check for optimality

if $\mathbf{c}^T\mathbf{x}^{(k+1)}/\mathbf{c}^T\mathbf{x}^{(0)} \leqq \varepsilon$ **then stop** (optimum obtained)

else $k=k+1$, **go to** 1-**endif**.

2.1. Numerical example and geometric representation

The easiest way to illustrate some of the major features of the Karmarkar algorithm is to use a simple numerical example.

Consider the problem

$$\begin{gathered}\text{Min } z=2x_1-x_2\\ \text{s.t. } 3x_1+x_2=4\\ x_1\geqq 0, \qquad x_2\geqq 0,\end{gathered}$$

whose optimum objective function value is $z^*=-4$. By introducing an additional variable x_3 we may write this as:

$$\begin{gathered}\text{Min } 2x_1-x_2-z^*x_3\\ \text{s.t. } 3x_1+x_2-4x_3=0:\Omega\\ x_1+x_2+x_3=1:\Sigma\\ x_1\geqq 0, \qquad x_2\geqq 0, \qquad x_3\geqq 0.\end{gathered}$$

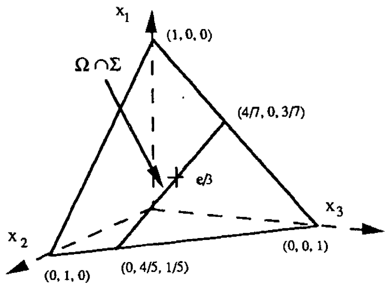


Figure 2.2. – Feasible Polytope $\Omega\cap\Sigma$.

The problem being in canonical form, the algorithm may be applied if a feasible interior point is available to start with. For this purpose, point $\mathbf{x}_0=\mathbf{e}/3=(1/3,\ 1/3,\ 1/3)^T$ is interior feasible , as it belongs to the line segment between points (0, 4/5, 1/5) and (4/7, 0,3/7) in $\Omega\cap\Sigma$ which is the feasible region. It is also the centre of the simplex Σ as depicted in figure 2.2.

The first iteration of the algorithm requires rescaling the feasible region, using the projective transformation $T(\mathbf{x})=\mathbf{x}'=D^{-1}\mathbf{x}/\mathbf{e}^T D^{-1}\mathbf{x}$, and its inverse $T^{-1}(\mathbf{x}')=\mathbf{x}=D\mathbf{x}'/\mathbf{e}^T D\mathbf{x}'$, where $D=\text{diag}(\mathbf{x}_0)=\text{diag}(1/3,\ 1/3,\ 1/3)$. However, $\mathbf{x}_0$ being already at the centre of the simplex, the transformation is equivalent to an identity, which leaves the region as in figure 2.2. The objective function, however, has changed. The transformed problem is

$$\begin{gathered}\text{Min } 2/3\,x_1'-1/3\,x_2'-(z^*/3)\,x_3'\\ \text{s.t. } x_1'+1/3\,x_2'-4/3\,x_3'=0:\Omega\\ x_1'+x_2'+x_3'=1:\Sigma\\ x_1'\geqq 0, \qquad x_2'\geqq 0, \qquad x_3'\geqq 0.\end{gathered}$$

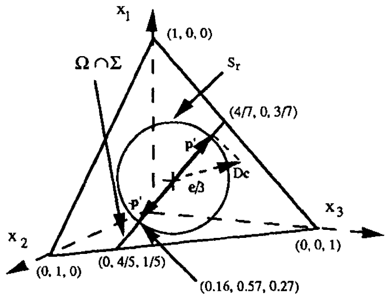


Figure 2.3. – A Step Along the Projected Gradient.

In step 2 of Karmarkar's algorithm the steepest descent direction in the transformed deasible region is found. The direction-$\mathbf{p}'$ is found by projecting the gradient $D\mathbf{c}$ onto the polytope $\Omega \cap \Sigma$, *i.e.* multiplying the projection matrix H with vector $D\mathbf{c}$, and considering the negative of this vector (see *fig.* 2.3). It is along the negative gradient that the objective function decreases most rapidly.

In step 4 a move in the direction-$\mathbf{p}'$ is made. However, to guarantee feasibility after the move, a sphere S_r of radius $\alpha[n(n-1)]^{-1/2}$ centered at $\mathbf{e}/3$ is inscribed inside the triangle Σ of above figures, and the minimization is over $\Omega \cap \Sigma \cap S_r$ which is a sphere but of lower dimension. All the points of this lower dimension sphere are feasible. In figure 2.3 it is the segment which is delimited by S_r in $\Omega \cap \Sigma$. Notice that $r=[n(n-1)]^{-1/2}$ is the radius of the largest sphere that can be inscribed in Σ. A fraction α of r is only taken to avoid infeasibility, with $0<\alpha<1$. The move is then of length αr, with $\alpha=0.9$ and results in point (0.16, 0.57, 0.27). The point is not optimal as it does not reduce the objective function to zero. Note that if the move was long enough, we would have reached the optimum solution, which is the end point (0, 4/5, 1/5) of segment $\Omega \cap \Sigma$. However, this is so because the feasible region is a segment and the solution is one of its two vertices. In higher dimensions it would not be so easy to identify the solution.

Having obtained a new point, we can proceed with the next iteration. The problem is first transformed into

$$\begin{aligned}
&\text{Min } 0.32x_1' - 0.57x_2' + 1.08x_3' \\
&\text{s.t. } 0.48x_1' + 0.57x_2' - 1.08x_3' = 0 : \Omega \\
&\qquad x_1' + x_2' + x_3' = 1 : \Sigma \\
&x_1' \geq 0, \qquad x_2' \geq 0, \qquad x_3' \geq 0.
\end{aligned}$$

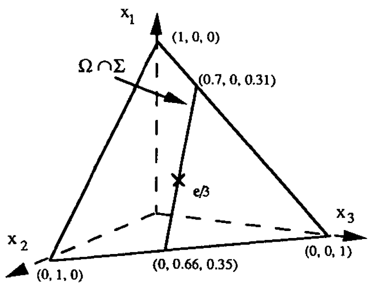


Figure 2.4. – **Rescaling of the Feasible Region.**

Again, the search direction is computed by multiplying the projection matrix H and the gradient of the objective function in above problem. The optimization process over $\Omega \cap \Sigma \cap S_r$ produces point (0.07, 0.59, 0.35) as depicted in figure 2.5. When transformed back to the space of the canonical form, point (0.11, 3.56, 1.0) is obtained which is close to the optimal solution $x^*=(0.0, 4.0, 1.0)$. The corresponding objective function is $z=0.67$, which is still much larger than zero. One more iteration is necessary to get a good approximation to the optimum solution.

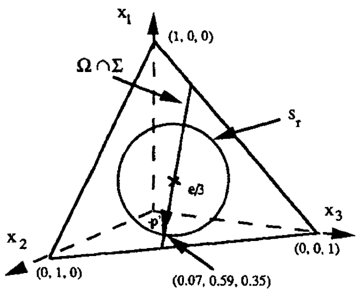


Figure 2.5. – Result of Iteration 2.

Many problems arose with the efficient implementation of the original Karmarkar algorithm including the choice of a more efficient step length in the direction p, relaxing the requirement that the optimum objective function be known in advance and the effective application of the algorithm to sparse problems. In the following sections we consider these problems in more detail.

3. EXPERIMENTAL APPROACH TO KARMARKAR'S ALGORITHM

An important characteristic of large linear programming problems, is sparsity. To achieve efficiency when solving these problems it is important to exploit it. In the following discussion we shall describe how a variant of the projective algorithm [Karmarkar, 1984*a*, 1984*b*] was implemented to solve Klee-Minty and Hilbert type problems as well as real world sparse LP problems. We shall examine the performance of the algorithm in conjunction with the form in which the problem is handled and the technology available for least squares problems. Issues related to the coding of the algorithm in Fortran 77 such as data structures and input data (MPS format) will be discussed.

Two implementations of the algorithm are considered: LPKAR 1 and LPKAR 2. In LPKAR 1 it is assumed that the optimum objective value z^* of the problem is *a priori* known while in LPKAR 2 the assumption is dropped. The canonical form in which the problem is handled differs for both cases. The first code works on a canonical form in which the objective function is included as a constraint with z^* as its right-hand side. It works in a single phase and provides only the primal solution. The second code, LPKAR 2 works on the canonical form described in Lustig (1985). It is a two-phase method and generates dual solutions.

Aspects of sparsity exploitation such as ordering and partitioning will be discussed and numerical results obtained using LPKAR 1 and LPKAR 2 will be presented.

3.1. A variant of the Karmarkar algorithm

The original algorithm of Karmarkar (1984 *a*, 1984 *b*) was implemented and did not perform as efficiency as expected. The difficulties encountered were partly due to the inflation of the problem size after primal-dual combination to put the problem in the required form, and also to ill-conditioning in the projection matrix which involves inverting a cross-product matrix of the form BB^T. However, this implementation provided valuable insights to the properties of the algorithm and its behaviour. The steplength, for instance does not have to be 1/4 as suggested by Karmarkar to insure convergence. Indeed, values closer to 1.0 and even larger, greatly improve the speed of convergence. We also noted that the number of iterations is generally low, which confirmed Karmarkar's claim. In the light of these observations and experience, we present a variant of the algorithm on which our codes were based. NB the prime notation used in the algorithm description and throughout this paper does not denote transpose but indicates the matrix or vector is modified in the way defined.

Algorithm 3.1

The following algorithm handles problems in standard form, *i.e.* $\{\mathbf{x}\in R^n \mid min\ \mathbf{c}^T\mathbf{x},\ A\mathbf{x}=\mathbf{b},\ \mathbf{x}\geqq 0\}$. Assume that an interior feasible point is at hand, then

1. Transform problem into the form

$$\begin{gathered}\min\ \mathbf{c}'^T\mathbf{x}'\\ \text{s.t. } A'\mathbf{x}'=0,\\ \mathbf{x}'\geqq 0,\end{gathered}$$

where $\mathbf{c}'^T=[\mathbf{c}^T,\ -z]$, $A'=[A,\ -\mathbf{b}]$ and $\mathbf{x}'^T=[\mathbf{x}^T,\ 1]$.

2. Initialization

$$k=0, \qquad \varepsilon=1.0\,\text{E-06},$$

$$z=M, \qquad \text{where } M \text{ is a large value}, \qquad D=\operatorname{diag}(\mathbf{x}^{(0)},\ 1).$$

```
3. **if** $\mathbf{c}'^T \mathbf{x}'^{(k)} < \varepsilon$ **stop**.
4. Compute $\mathbf{y} = (DA'^T)^{\dagger} D \mathbf{c}'$.
5. Compute $\mathbf{p} = D\mathbf{c}' - (DA'^T)\mathbf{y} - (\mathbf{c}^T \mathbf{x}^{(k)}/n)\,\mathbf{e}$.
6. Normalize $\mathbf{p}$, i.e. $\mathbf{p}' = \mathbf{p}/\|\mathbf{p}\|$.
7. $\xi^{(k+1)} = \mathbf{e} - \alpha\,\mathbf{p}'$, where $\alpha$ is the steplength.
8. Compute $\mathbf{x}'^{(k+1)} = D\xi^{(k+1)}/\mathbf{e}^T D\xi^{(k+1)}$.
9. Compute $\mathbf{x}'^{(k+1)} := \mathbf{x}'^{(k+1)}/x'^{(k+1)}_{n+1}$.
10. $D = \mathrm{diag}(\mathbf{x}'^{(k+1)})$, $\mathbf{c}'^T = [\mathbf{c}^T, -\mathbf{c}^T \mathbf{x}^{(k+1)}]$, $k = k+1$, **go to** 3.
```

Note that $\mathbf{x}$ is made up with the n first entries of $\mathbf{x}'$. Also $\dagger$ denotes the pseudo-inverse of the matrix and T denotes the transpose of the matrix or vector. The Nag subroutine F01 BLF can, for, example be used to compute the required pseudo-inverse.

Algorithm 3.1 differs from the original Karmarkar's algorithm and the variant described by Lustig (1985) in the way the projection matrix and the search direction are computed. This approach is more suitable as the sparsity of the original problem is only slightly disturbed by adjoining the column correspong to the right-hand side. When optimum objective value z^* is available, it can be shown that Algorithm 3.1 retains the polynomial complexity of Karmarkar's algorithm. On the other hand, if z^* is not supplied, then updates of z, *i.e.* $\mathbf{c}^T \mathbf{x}^{(k+1)}$, after each iteration can be used instead. However, while it is possible to establish that $\mathbf{c}^T \mathbf{x}^{(k+1)} < \mathbf{c}^T \mathbf{x}^{(k)}$, which implies $\mathbf{p}'$ is a descent direction, it is difficult to show whether the algorithm is polynomial in time. To make sure that Algorithm 3.1 has polynomial complexity while dealing with unknown optimum objective value, the strategy that finds ever better lower bounds on z^*, described by Todd and Burrell (1986) and Ye and Kojima (1987), must be used.

4. IMPLEMENTATIONS OF ALGORITHM 3.1

LPKAR 1 and LPKAR 2 are two different ways of applying Algorithm 3.1 to a linear programming problem depending on assumptions made and information available about the problem. In LPKAR 1 sparsity exploitation is the central issue. This will involve symbolic factorization, ordering and updating techniques. In LPKAR 2 we investigate the possibility of solving LP problems without supplying z^* and by using the Moore-Penrose pseudo-inverse [Ben-Israel & Greville, 1974] to solve the least squares problem of step 4 in Algorithm 3.1. First, let us look at the form under which the problem is handled by LPKAR 1.

Assuming that z^* is available, it is possible to transform the original problem in standard form into the following equivalent form accepted by Algorithm 3.1.

$$\left.\begin{array}{l} \min \lambda \\ \text{s.t. } A\mathbf{x}-\mathbf{b}-(A\mathbf{e}-\mathbf{b})\lambda=0 \\ \mathbf{c}^T\mathbf{x}-z^*=0 \\ \mathbf{x},\ \lambda \geqq 0, \end{array}\right\} \quad (4.1)$$

where λ is an artificial variable.

This problem is in R^{n+2} and admits $\mathbf{e}_{n+2}$ as an interior feasible point. Algorithm 3.1 is readily applicable. One advantage this form of the problem offers is that the optimum solution is obtained in one phase. Indeed, when λ is reduced to zero, the resulting point $\mathbf{x}^*$ satisfies the constraints in the original space, *i.e.* R^n, and also the extra constraint $\mathbf{c}^T\mathbf{x}-z^*=0$. It follows that $\mathbf{x}^*$ is optimum solution.

The other advantage of above canonical form is that the objective vector is zero except for one entry corresponding to the artificial variable λ. In LPKAR 1 this sparsity is used to reduce the work in an iteration of Algorithm 3.1. The way this is brought to effect will be shown later. Note that LPKAR 1 is a primal only method as with the original algorithm of Karmarkar. Results are obtained using LPKAR 2, later in this paper, giving the dual and primal solutions which in addition does not require advance knowledge of the optimum value of the objective function.

4.1. Details of LPKAR1

LPKAR1 uses Cholesky method to deal with the least squares problem of step 4 in Algorithm 3.1 augmented with sparsity preservation steps comprising the Nested Dissection Ordering algorithm of George [George & Liu, 1981], and a version of the updating technique for least squares of Heath (1984). Symbolic factorization is also used to set up appropriate data structures. With these steps added, Algorithm 3.1 can be described as follows.

Algorithm 4.1

Assume that a feasible point $\mathbf{x}^{(0)}$ is available and that the problem is in the canonical form (4.1) accepted by Algorithm 3.1.

1. Initialization

$$k=0, \qquad \varepsilon=1.0\,\mathrm{E}-06,$$

$$z=M, \qquad \text{where} \quad M \text{ is a large value}, \qquad D=\operatorname{diag}(\mathbf{x}^{(0)}, 1).$$

2. **if** $\mathbf{c}'^{T}\mathbf{x}'^{(k)}<\varepsilon$ **stop**

3. Compute $\mathbf{y}$ as follows

(*a*) Remove the full rows of matrix DA'^{T}.

(*b*) Find symbolic representation or adjacency structure of $A'D^2A'^{T}$.

(*c*) Find a permutation matrix P using the Nested Dissection Ordering Algorithm.

(*d*) Find a symbolic factorization of $PA'D^2A'^{T}P^{T}$, *i.e.* find the non-zero structure of the Cholesky factor L of the cross-product.

(*e*) Fill the structure with the actual numerical values by applying Cholesky or Givens method.

(*f*) Apply a forward and a back substitution to get the solution $\mathbf{y}'$ to the incomplete least squares problem.

(*g*) Apply inverse ordering to get incomplete solution in the original ordering.

(*h*) Add effect of the removed rows to the solution $\mathbf{y}'$ by updating it using the algorithm of Heath (1984), resulting in $\mathbf{y}$.

4. Compute $\mathbf{p}=D\mathbf{c}'-(DA'^{T})\mathbf{y}-(\mathbf{c}^{T}\mathbf{x}^{(k)}/n)\mathbf{e}$.

5. Normalize $\mathbf{p}$, *i.e.* $\mathbf{p}'=\mathbf{p}/\|\mathbf{p}\|$.

6. $\xi^{(k+1)}=\mathbf{e}-\alpha\mathbf{p}'$, where α is the steplength.

7. Compute $\mathbf{x}'^{(k+1)}=D\xi^{(k+1)}/\mathbf{e}^{T}D\xi^{(k+1)}$.

8. Compute $\mathbf{x}'^{(k+1)}/x'^{(k+1)}_{n+1}$.

9. $D=\operatorname{diag}(\underset{\bullet}{\mathbf{x}}'^{(k+1)})$, $\mathbf{c}'^{T}=[\mathbf{c}^{T}, -\mathbf{c}^{T}\mathbf{x}^{(k+1)}]$, $k=k+1$, **go to** 2.

Note that this extended version of Algorithm 3.1 may be simplified due to the very sparse cost vector of problem (4.1).

4.1.1. *Adjacency structure of* $A'D^2A'^{T}$

Ordering algorithms are graph techniques and are known to be sensitive to the way the graphs are represented. In our case, to proceed with the reordering of the cross-product $A'D^2A'^{T}$ and set up the data strutures for the Cholesky factor, it is essential to efficiently store its nonzero structure and retrieve adjacency relations. Thus, the adjacency structure of a matrix is the representation of its graph.

Let $G(x, E)$ be a graph with N nodes. The adjacency list of a node $x \in X$ is a list containing all adjacent nodes to x and the structure of G is the set of such lists for all its nodes. The implementation of the structure is done by storing the adjacency lists sequentially in a one-dimensional array *ADJNCY* along with an index vector *XADJ* of length $N+1$ containing pointers to the beginning of the lists in *ADJNCY* (see *fig.* 4.1). The extra entry *XADJ* $(N+1)$ points to the next available location in *ADJNCY* [George & Liu, 1981].

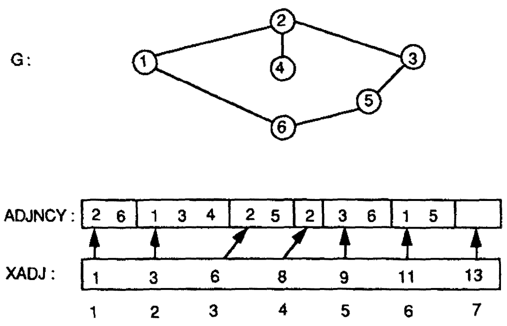


Figure 4.1. – Adjacency Structure of Graph G.

The attractive feature of this approach is that the structure of $(A' D)(A' D)^T$ is found without explicitly forming the cross-product.

4.1.2. *Symbolic factorization and storage scheme*

After applying the nested dissection ordering algorithm, a permutation matrix P is returned which will help reduce fill-in during the factorization process of $PA' D^2 A'^T P^T$. We should note, however, that the nested dissection algorithm has been proved theoretically efficien only for planar network type adjacency structures.

Before proceeding with the actual numerical factorization, a simulation of it, or symbolic factorization is carried out to set up the data structures to contain the Cholesky factor in sparse form. The advantage of this approach is that the data structures are static; thus they are set up once and for all, as the structure of the matrix does not change from iteraction to iteration. Not that at this stage the numerical vaues of the Cholesky factor are not explicitly computed.

The data structures returned by the symbolic factorization are presented in a sparse storage scheme known as the compressed scheme of Sherman, cited in [George & Liu, 1981]. The scheme has a main one-dimensional storage array *LNZ* which will contain all nonzero entries in the lower triangular factor of $PA' D^2 A'^T P^T$ column-wise, an *INTEGER* vector *NZSUB* which will hold the row subscripts of the nonzeros, and an index vector *XLNZ* whose entries are pointers to the beginning of nonzeros in each column in *LNZ*. In addition, an index vector *XNZSUB* is also use to hold pointers to the start of row subscripts in *NZSUB* for each column. The diagonal elements are stored separately in vector *DIAG*.

4.2. Input data for codes of algorithm 3.1

Real world problems usually are stored in *MPS* format which is standard in industry. The format, mainly, consists of three sections: constraints type, constraints entries stored column-wise including the cost vector and the right-hand side. Other sections may be added such as bounds on variables and free constraints.

Example 1:

```
NAME          PROB1
ROWS
 N  FOB00001
 G  ROW00001
 G  ROW00002
 G  ROW00003
COLUMNS
    COL00001  FOB00001     -5.000000  ROW00001      -2.000000
    COL00001  ROW00002     -4.000000  ROW00003      -3.000000
    COL00002  FOB00001     -4.000000  ROW00001      -3.000000
    COL00002  ROW00002     -1.000000  ROW00003      -4.000000
    COL00003  FOB00001     -3.000000  ROW00001      -1.000000
    COL00003  ROW00002     -2.000000  ROW00003      -2.000000
RHS
    RHS       ROW00001     -5.000000  ROW00002     -11.000000
    RHS       ROW00003     -8.000000
ENDATA
```

The problem under *MPS* format is read into a one-dimensional array *ALIST* of length *NZ*, which is a column-wise storage of the problem matrix. Slack variables are added according to the type of constraints encountered as well as the two columns, $-\mathbf{b}$ and $-(A\mathbf{e}-\mathbf{b})$ required by the canonical form. *ALIST* is accompanied with two *INTEGER* vectors, *ICOL* and *IT*, with lengths *NZ* and $N+1$, N being the number of total variables in the canonical form. *ICOL* contains the row subscript of each nonzero in *ALIST*, while *IT* contains pointers to the beginning of each column.

Our implementation requires that we repeatedly form the matrix $A' D^2 A'^T$ as D changes from iteration to iteration. Thus, to avoid searching for the rows of A' in *ALIST*, we preferred to store the matrix in row-wise form.

This may seem inefficient regarding space, however, it makes sense from the time point of view. Consequently, we have another trio of vectors $RA(NZ)$, $IA(NZ)$ and $NA(M+1)$ containing A' row-wise. These arrays are filled in once only by performing a fast sparse-matrix transposition after *ALIST*, *ICOL* and *IT* have been constructed.

4.3. Computational experience

LPKAR1 was tested on the problems listed in table I whose origins are as follows:

Chvtl1 and Chvtl2, respectively, are a farm planning *LP* model and a case study in forestry described in Chvátal (1983). Alfaut and RandD were borrowed from *ICL LP*3 manual (1973). The remaining problems are standard test problems supplied to us by Dr. Etienne Loute of the Catholic University of Louvain, Belgium and described in Ho and Loute (1980).

TABLE I

Test Problems Statistics.

Problem	Original Form		Canonical Form				z*
	Rows	Cols	Rows	Cols	Nonzeros	Density %	
Chvtl1	16	11	17	28	142	29.83	-14021.04
Chvtl2	17	13	18	32	114	19.79	-273382.1
Alfaut	38	33	39	72	301	10.72	-12233742
RandD	39	15	40	56	396	17.68	-9474.4845
Scsd1	77	760	78	762	3268	5.43	8.666667
Scagr7	129	140	130	187	782	3.22	-2331390
Scsd6	147	1350	148	1352	5824	2.91	50.50000
Sc205	205	203	206	319	911	1.39	-52.20206
Sctap1	300	480	301	662	2688	1.35	1412.250
Scfxm1	330	457	331	602	3203	1.61	18416.76
Scagr25	471	500	472	673	2852	0.90	-14753433

The results reported below (table II through 5) concern the performance of Algorithm 3.1 in conjunction with the nested dissection ordering algorithm and the updating algorithm for least squares. Four versions of LPKAR1 were run on all the test problems. The versions differ in the ways sparsity is exploited. Four cases arise:

Case 1 : Ordering and partitioning were not implemented in LPKAR1 (table II).

TABLE II

*Performance of LPKAR*1 (*Case* 1).

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.23
Chvtl2	153	9	0.20
Alfaut	741	16	1.94
RandD	780	14	2.32
Scsd1	3003	13	66.21
Scagr7	7232	20	35.25
Scsd6	10878	15	264.41
Sc205	20914	23	157.25
Sctap1	45150	28	669.00
Scfxm1	54519	25	797.36
Scagr25	94244	29	1948.77

TABLE III

*Performance of LPKAR*1 (*Case* 2).

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.26
Chvtl2	153	9	0.23
Alfaut	741	16	2.12
RandD	780	14	2.26
Scsd1	1390	12	64.29
Scagr7	6230	18	30.25
Scsd6	3167	14	240.45
Sc205	20317	22	157.95
Sctap1	45150	27	672.69
Scfxm1	54047	25	839.53
Scagr25	79994	25	1404.74

Case 1 : The nested dissection ordering algorithm was implemented, but no partitioning was considered (table III).

Case 3 : The partitioning or updating Algorithm 1 was implemented, but no ordering was performed (table IV).

Case 4 : Both ordering and partitioning were implemented in LPKAR1 (table V).

Beside the *CPU* time (in sec.) and the number of iterations taken by the four versions of LPKAR1 on all the test problems, a column containing the

TABLE IV

*Performance of LPKAR*1 (*Case* 3).

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.24
Chvtl2	124	9	0.16
Alfaut	196	16	1.16
RandD	771	15	2.13
Scsd1	1408	13	42.05
Scagr7	1250	18	9.56
Scsd6	2779	14	217.32
Sc205	1574	22	17.92
Sctap1	8286	28	153.56
Scfxm1	12075	25	204.31
Scagr25	4922	28	177.99

TABLE V

*Performance of LPKAR*1 (*Case* 4).

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.24
Chvtl2	61	9	0.14
Alfaut	104	17	1.25
RandD	690	14	1.88
Scsd1	1393	12	49.52
Scagr7	1116	19	10.59
Scsd6	3119	14	219.35
Sc205	1507	22	19.39
Sctap1	3736	27	128.90
Scfxm1	6812	26	180.13
Scagr25	4848	25	170.59

number of nonzeros in the Cholesky factor for each problem is included. This column, with the heading « R Nonzeros », clearly shows advantages and disadvantages of both ordering and updating techniques. Please note that in tables 2 to 5 of this section the small differences in the number of iterations taken to solve some of the problems are due to rounding errors.

In these experiments, the potential function as well as the objective function λ of the canonical form (4.1) are monitored for some of the problems of table I. These functions are represented in the graphs below. The potential function is the logarithmic function of Karmarkar (1984*a*, 1984*b*).

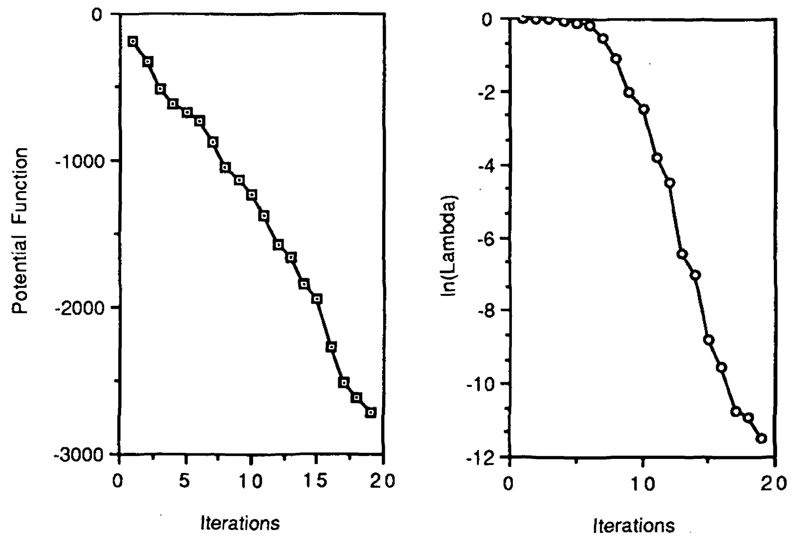


Figure 4.2. — Decrease in the Potential and Objective Functions for Problem Scagr7.

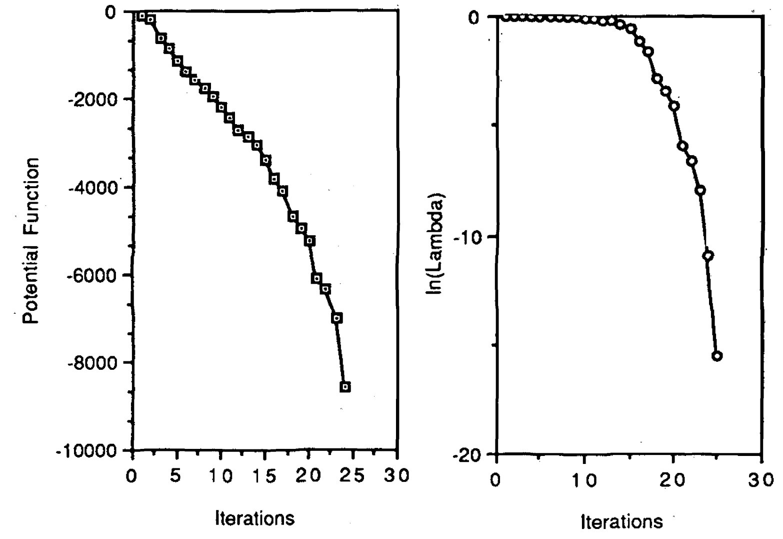


Figure 4.3. — Decrease in the Potential and Objective Functions for Problem Scagr25.

4.3.1. *Hilbert-type LP problems*

A version LPKAR1 which does not take account of sparsity was tested on a set of *LP* problems whose constraints matrix is based on the Hilbert

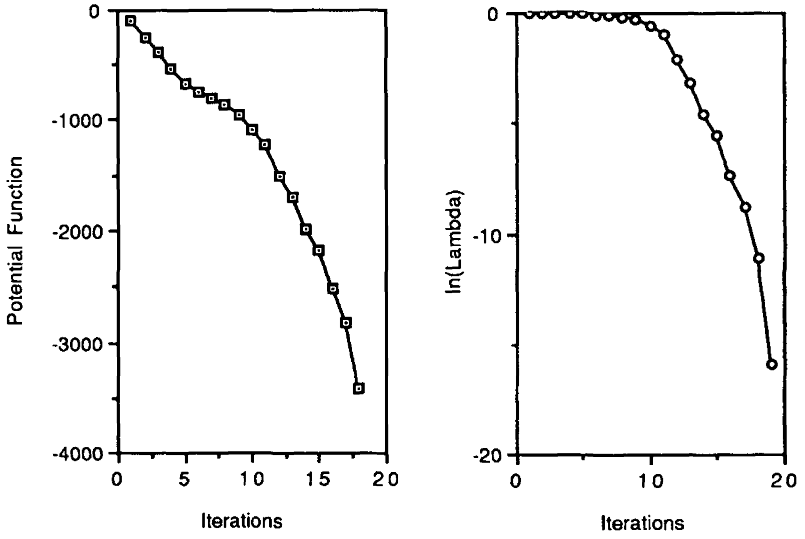


Figure 4.4. — Decrease in the Potential and Objective Functions for Problem Sc205.

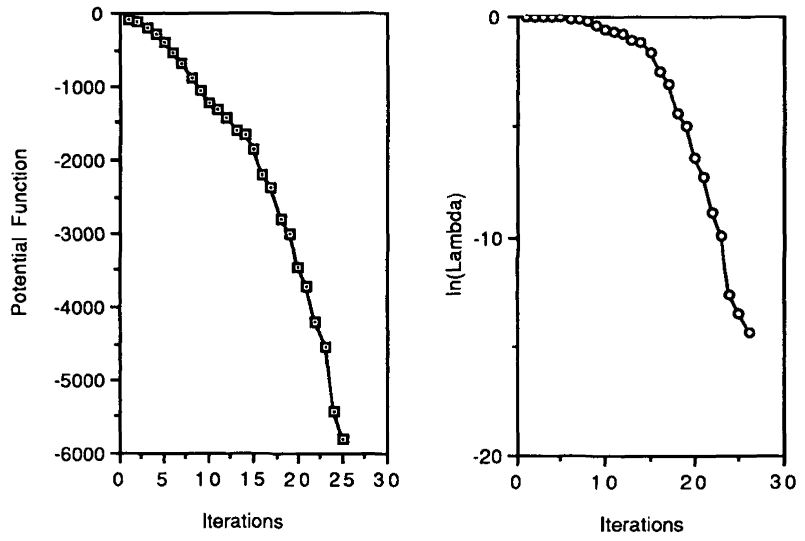


Figure 4.5. — Decrease in the Potential and Objective Functions for Problem Scfxm1.

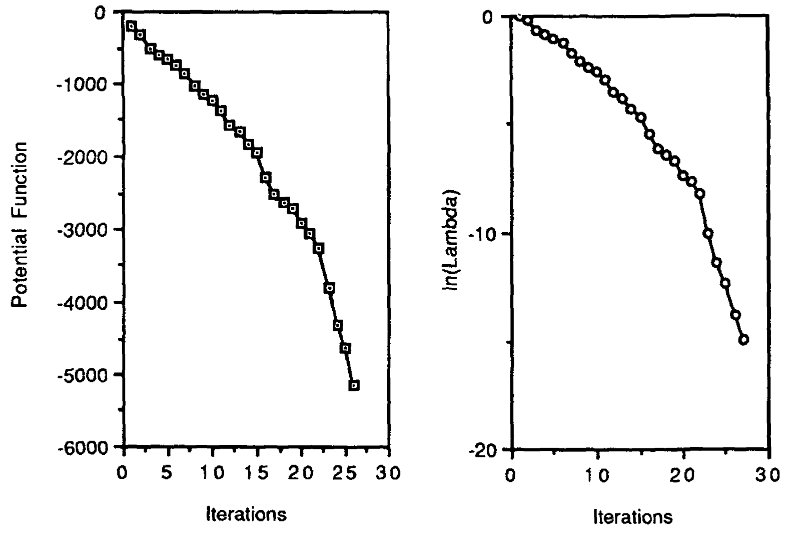


Figure 4.6. — Decrease in th Potential and Objective Functions for Problem Sctapl.

matrix. They are of the form

$$\begin{aligned} &\min \mathbf{c}^T \mathbf{x} \\ &\text{s.t. } A\mathbf{x} \geqq \mathbf{b}, \\ &\mathbf{x} \geqq 0, \end{aligned}$$

where $\mathbf{x} \in R^n$, $A \in R^{n \times n}$, $\mathbf{c} \in R^n$ and $\mathbf{b} \in R^n$. Matrix A has entries $[a_{ij}] = [1/(i+j)]$, for $i = 1, \ldots, n$ and $j = 1, \ldots, n$. The RHS is given by

$$b_i = \sum_{j=1}^{n} \frac{1}{i+j}, \qquad i = 1, 2, \ldots, n.$$

The cost vector is given by

$$c_i = \frac{2}{i+1} + \sum_{j=1}^{n} \frac{1}{i+j}, \qquad i = 1, 2, \ldots, n.$$

The primal optimum solution to these problems is $\mathbf{x}^* = (1, 1, \ldots, 1)^T$. Problems with $n = 4$, 6, 10, 15, 20, 25 and 30 were solved and the results depicted in figure 4.7.

As one would expect, the number of iterations is approximately the same for problems with $n > 6$. Around iteration 16, the potential function levels out and shows hardly any noticeable improvement.

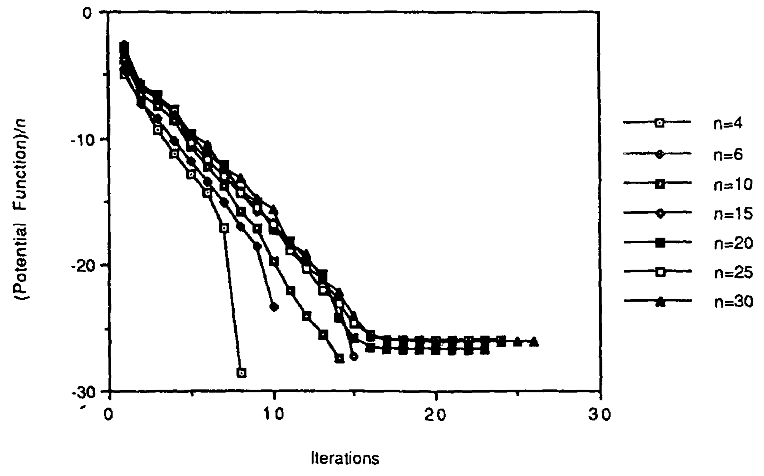


Figure 4.7. — Results from LPKAR1 on Hilbert-Type Problems.

4.3.2. *Klee-Minty problems*

The class of problems originally proposed by Klee and Minty (1972) is well known as linear programming problems with n variables for which the simplex method with various pivot rules takes an exponential, in n, number of pivots to reach the optimum. The following form due to Avis and Châtal (1978) is considered in our experiments, as well as in [Iri & Imai, 1986].

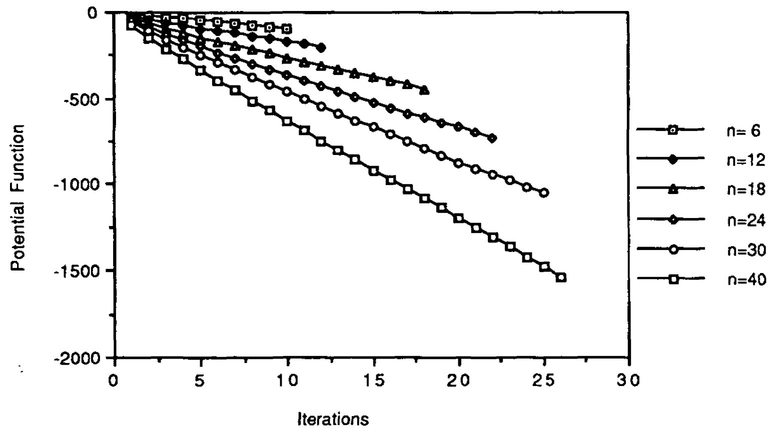
$$\max \sum_{j=1}^{n} \mu^{n-j} x_j,$$

$$\text{s.t.} \sum_{j=1}^{n} \mu^{i-j} x_j + x_i \leqq 1, \qquad (i=1, \ldots, n),$$

$$x_j \geqq 0, \qquad (j=1, \ldots, n),$$

where $0<\mu<0.5$. The optimum solution of this problem is $x_j=0$ $(j=1, \ldots, n-1)$ and $x_n=1$. We performed experiments for the cases with $\mu=0.4$ and $n=6$, 12, 18, 24, 30 and 40. The results are shown in figure 4.8.

Although the iteration count is still low for the Klee-Minty problem, the number of iterations seems to grow slightly with the size of the problem. But it is nothing like the simplex method. For the Klee-Minty problem of order 40, for instance, the standard simplex would take approximately 10^{12} iterations, as compared to 27 iterations the Karmarkar algorithm takes to find the optimum solution. The growth with the size is logarithmic and not exponential.



5. APPLYING ALGORITHM 3.1 WHEN z* IS NOT *A PRIORI* KNOWN

This feature is implemented in the program LPKAR2. However before dealing with the unknown optimum objective value, it is necessary to find a starting feasible point. This can be done by solving a feasibility problem, otherwise known as the Phase 1 problem. This problem is similar to the one solved by LPKAR1. The unknown optimum objective value is dealt with by updating the initial value of z in Algorithm 3.1 with $\mathbf{c}^T \mathbf{x}^{(k)}$ after iteration k. This approach is known as the cutting objective function method. Algorithm 3.1 with the cutting objective is a primal-dual algorithm. The vector $\mathbf{y}$ computed in step 4 is dual feasible. At the end of Phase 2, $\mathbf{y}$ is the true dual optimum solution if the problem is nondegenerate, *i.e.* $\mathbf{x}^*$ has at least $m+1$ positive entries, where m is the number of constraints. Otherwise the dual solution is not unique and a number of alternative dual feasible solutions provide the same optimum objective function.

LPKAR2 is a FORTRAN77 code for this algorithm. Step 4 of Algorithm 3.1 is carried out using the Nag subroutine F01BLF for computing the pseudoinverse.

The code was run on a subset of the problems listed in table 1. The results of these runs are given below. For each problem 5 columns were produced, which respectively are: the iteration number, the optimum step α taken at that iteration, the primal objective value, a lower bound on the dual objective function and its value. The bank entries to the last two columns correspond to Phase 1 iterations in which an interior feasible point is found.

Problem Name: RandD

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	6.441861428	0.6531709614		
2	4.559867337	0.820182E-01		
3	1.445073937	0.771727E-03		
4	4.621143626	-8884.832099	-16428.78711	-16175.50098
5	3.841302410	-9320.974037	-13733.94824	-13591.17871
6	3.682589382	-9412.218261	-11239.68359	-11188.44922
7	3.033384722	-9436.795379	-11637.96973	-11584.12500
8	2.566412224	-9450.617668	-.100000E+21	-.975998E+20
9	2.358197628	-9463.677588	-9715.130859	-9708.828125
10	1.669588275	-9466.686918	-9491.047852	-9490.426758
11	2.635119288	-9467.923004	-9471.314453	-9471.197266
12	1.636255498	-9468.274640	-9468.140625	-9468.111328
13	6.255412489	-9472.807924	-9467.330078	-9467.324219
14	2.247326661	-9477.173856	-9565.544922	-9563.179688
15	6.222729319	-9475.909702	-9474.494141	-9474.494141

Problem Name : Chvtl1

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	2.865809331	0.9934241435		
2	3.966572662	0.7941611308		
3	2.050811475	0.290249E-01		
4	1.008751969	0.287112E-03		
5	9.301639104	-11483.16638	-19023.27539	-18619.72656
6	2.452507189	-13027.91157	-17058.04688	-16811.80273
7	1.972840325	-13286.31012	-17406.45898	-17182.81641
8	14.74104330	-13651.34425	-16813.26172	-16627.70703
9	2.797717475	-13771.54418	-16001.35547	-15872.47559
10	4.927777042	-13778.51607	-16125.33691	-15988.55273
11	5.805054359	-13779.43317	-16142.97949	-16004.45898
12	21.45803992	-13791.03023	-16141.19727	-16002.74707
13	10.02461707	-13878.35165	-15772.87598	-15660.45605
14	4.149567088	-14012.74683	-14595.31836	-14553.69141
15	20.14787443	-14015.04564	-14152.19531	-14144.03809
16	20.12815984	-14015.08041	-14123.12012	-14116.76367
17	25.84468262	-14015.08173	-14122.71289	-14116.38184
18	30.98095870	-14015.08194	-14122.70898	-14116.37793
19	108.0583358	-14015.07551	-14122.70898	-14116.37793
20	4.082054439	-14015.33463	-14122.70898	-14116.37793
21	7700.428266	-14015.08186	-14122.70801	-14116.37695
22	4.081874178	-14015.08029	-14122.70898	-14116.37793
23	24228.34324	-14015.05874	-14122.70703	-14116.37598
24	2371190485.	-14015.22800	-14122.57227	-14116.25000
25	70.28025742	-14021.00474	-14117.38281	-14111.38184
26	44.80198279	-14021.03772	-14021.04102	-14021.04102

Problem Name : Chvtl2

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	4.403545515	0.9944982521		
2	2.450272242	0.5077499693		
3	1.131387576	0.965917E-02		
4	1.005887171	0.944669E-04		
5	2.147604523	-228535.2657	-642805.9375	-626204.3125
6	3.016425300	-250208.7442	-317029.2813	-314349.6250
7	2.819401851	-263412.8438	-312417.2500	-310173.6875
8	3.022792842	-267849.2021	-315956.2188	-313710.1875
9	4.931871498	-269401.4924	-315477.8438	-313141.0000
10	6.209180867	-269913.1435	-314532.0625	-312117.5000
11	9.077869172	-273307.8685	-309012.2813	-306870.6875
12	29.57530165	-273377.0912	-273385.5000	-273385.3125
13	29.02232090	-273381.8568	-273382.0000	-273382.0000

The stopping criterion used is based on the gap between the primal objective and its lower bound. Steplength α is computed at each iteration using the blocking variable technique described in Lustig (1985). Comments on the following results can be found in the conclusion.

6. COMPARATIVE RESULTS BETWEEN LPKAR1 (Case 3) and LINDO

LINDO (Linear INteractive Discrete Optimizer), [Schrage, 1983], is a commercial package which does Linear as well as Integer and Quadratic

Problem Name : Alfaut

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	5.020584304	0.9987008107		
2	5.016395983	0.7061022659		
3	1.224532672	0.226881E-01		
4	0.997201850	0.228803E-03		
5	7.701221428	-9256687.028	-36732648.00	-36303604.00
6	3.943619602	-10490237.48	-31063900.00	-30724012.00
7	2.934036697	-11106791.07	-16443347.00	-16347721.00
8	3.248521773	-11658852.95	-14401056.00	-14348024.00
9	7.467750399	-12046247.56	-13226854.00	-13200235.00
10	4.741095164	-12171514.61	-12525954.00	-12517692.00
11	5.997308168	-12218466.46	-12263207.00	-12262387.00
12	8.387838113	-12230475.94	-12236620.00	-12236546.00
13	13.59068778	-12233368.10	-12234132.00	-12234121.00
14	27.19466703	-12233714.86	-12233751.00	-12233751.00
15	65.21966319	-12233741.39	-12233742.00	-12233742.00

Problem Name : Scagr7

```
ITERAT.        α            PRIMAL          L.BOUND          DUAL
-------       --            ------          -------          ----

   1     7.405831680   0.9981415303
   2     6.199722355   0.9863109023
   3     3.120554655   0.9243653430
   4     1.987366289   0.988973E-01
   5     .9975537117   0.108856E-02
   6     9.235479623   -2023372.180   -21206690.00    -21084080.00
   7     3.001158895   -2171044.627   -10496797.00    -10442040.00
   8     3.235121117   -2250286.867   -3377933.750    -3370697.250
   9     4.528271088   -2293157.895   -2505680.500    -2504344.000
  10     6.289297716   -2319053.392   -2383567.750    -2383144.500
  11     10.90774247   -2328609.473   -2354395.750    -2354224.000
  12     7.018505189   -2329835.032   -2335393.000    -2335359.000
  13     9.477241468   -2330885.659   -2334042.250    -2334019.500
  14     7.012442289   -2331150.209   -2333566.750    -2333549.250
  15     8.106021664   -2331280.375   -2332203.250    -2332196.750
  16     7.107331053   -2331328.956   -2331503.750    -2331502.750
  17     18.04304854   -2331369.400   -2331467.000    -2331466.500
  18     14.72354161   -2331381.338   -2331431.750    -2331431.500
  19     21.64842011   -2331386.993   -2331391.500    -2331391.500
  20     109.6028688   -2331389.556   -2331389.750    -2331389.750
```

Programming. It is available on Aston University's VAX 11/750 computer. To have an idea about the performance of our codes, we ran a version LPKAR1 (case 3) and LINDO on nine of the test problems given in table I. The results are recorded in table VI.

From the iteraction count point of view, LPKAR1 is superior to LINDO except on the small problems Chvtl1 and Chvtl2. However, LINDO requires less *CPU* time to solve all the problems.

TABLE VI

Comparative Results: LPKAR1 (Case 3) v LINDO.

	LPKAR1 (Case 3)		LINDO	
Problem	CPU(s)	IT	CPU(s)	IT
Chvtl1	5.79	15	3.69	11
Chvtl2	5.60	14	3.48	9
Alfaut	13.30	15	5.91	43
Scsd1	428.62	12	84.23	454
Scagr7	81.18	18	26.89	213
Sc205	148.73	21	54.89	207
Sctap1	1171.95	26	89.61	412
Scfxm1	1614.57	24	191.56	654
Scagr25	1514.95	27	377.13	1284

Note that the difference in *CPU* times required by LPKAR1 (Case 1) given in table IV and in the above table, is due to the computers used; the results of table IV were obtained on a VAX 8650 machine (6.5 mips), while the above results were obtained on a VAX 11/750 machine (0.7 mips).

7. CONCLUSION

Throughout these experiments, it is confirmed that Karmarkar's algorithm preserves its attractive features on various types of problems especially the real world ones listed in table I. These features, namely, are its low iteration count (logarithmic in the size of the problem) and its acceptance and use of the duality aspects of linear programming. Althrough the work in an iteration of the algorithm is substantially higher than that of the simplex [Tomlin, 185], it may be effectively reduced when existent sparsity techniques, such as ordering and patritioning, are used. In this way, large real world *LP* problems can be solved in realistic times as shown in table II through 5. The dependence of the performance of the algorithm on least squares techniques [Lindfield & Salhi, 1987] is also shown in those tables. This may be held a against the algorithm. However, any improvement in the solution of the least squares problem can readily be used in Karmarkar's algorithm.

The lower bound returned by LPKAR2 at iteration 8 for problem RandD is a pre-set value returned when the procedure fails to converge. The algorithm, nevertheless, recovers after this failure to converge to an acceptable solution. The problem of how accurate the solutions are is resolved by comparing the results with the true objective function values given in table I. The relative accuracy of the values achieved by the primal solutions and optimum objective values for all the problems is 10^{-6}.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous referees for their helpful comments.

REFERENCES

D. Avis and V. Chvátal, 1978, Notes on Bland's Pivoting Rule, *Math. Programming*, 8, pp. 24-34.

A. Ben-Israel and T. N. E. Greville, Generalized Inverses, *J. Willey and Sons*, 1974

A. Charnes, T. Song and M. Wolfe, An explicite Solution Sequence and Convergence of Karmarkar's Algorithm, Research Report CCS 501, Center for Cybernetic Studies, College of Business Administration 5.202, the University of Texas at Austin, Texas 78712-1177, U.S.A., 1984.

V. Chvâtal, Linear Programming, *W. H. Freeman & Co*, U.S.A., 1983.

A. George and J. W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Inc., *Englewood Cliffs*, NJ 07632, 1981.

M. T. Heath, Numerical Methods for Large Sparse Linear Least Squares Problems, *S.I.A.M. J. Sci. Stat. Comp.*, 1984, 4, (3), pp. 497-513.

J. K. Ho and E. Loute, A set of Staircase Linear Programming Test Problems, *Math. Programming*, 1980, 20, pp. 245-250.

M. Hiri and H. Imai, A Multiplicativ Barrier Function Method for Linear Programming, *Algorithmica*, 1986, 1, pp. 455-482.

N. Karmarkar, A New Polynomial-Time Algorithm for Linear Programming, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, 1984 *a*, pp. 302-311, Washington D.C.

N. Karmarkar, A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica*, 1984 *b*, A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica*, 4, (4), pp. 373-395.

V. Keel and G. J. Minty, How Good Is the Simplex Algorithm?, in *Inequalities III*, O. Shisha ed., Academic Press, N. Y., 1972, pp. 159-179.

G. R. Lindfield and A. Salhi, A Comparative Study of the Performance and Implementation of the Karmarkar Algorithm, presented at the *Martin Beale Memorial Symposium*, 6-8 July, 1987, The Royal Society, London.

I. J. Lusting, A Practical Approach to Karmarkar's Algorithm, TR SOL 85-5, Department of Operations Research, Stanford University, Stanford, CA 94305, 1985.

L. E. Schrage, *User's Manual for LINDO*, University of Chicago, U.S.A., 1983.

M. J. Todd and B. P. Burrell, An Extension of Karmarkar's Algorithm for Linear Programming Using Dual Variables, *Algorithmica*, 1986, 1, pp. 409-424.

J. A. Tomlin, An Experimental Approach to Karmarkar's Projective Methods for Linear Programming, *Proceedings of Symposium on Karmarkar's and Related Algorithms for Linear Programming*, organized by IMA, held on May the 7th 1985 at the Geological Society, Burlington House, Piccadilly, London, 1985.

Y. Ye and M. Kojima, Recovering Optimal Dual Solutions in Karmarkar's Algorithm for Linear Programming, *Math. Programming*, 39, (3), pp. 305-317.